Journal of Symbolic Computation 36 (2003) 271–285

Journal of Symbolic Computation

www.elsevier.com/locate/jsc

Computer proofs about finite and regular sets: the unifying concept of subvariance

Johan Gijsbertus Frederik Belinfante*

Georgia Institute of Technology, School of Mathematics, Atlanta, GA 30332-0160, USA

Received 27 September 2000; accepted 5 May 2001

Abstract

The work described here is a part of ongoing efforts to construct a set-theoretic framework that is convenient for automated reasoning in mathematics within first order logic. The specific topic in focus here is the theory of invariant and subvariant sets, which permits the development of a unified theory of regular and finite sets. Appendices are included listing theorems involving the axiom of regularity, the classes `REGULAR` and `FINITE` of regular sets and finite sets, respectively, as well as general theorems about invariant and subvariant subsets, all proved using McCune's automated reasoning program `Otter`.

Keywords: Finite; Regularity; Subvariance

1. Introduction

Computer-assisted proofs of elementary theorems of ordinal number theory were obtained recently (Belinfante, 1999a,b), using McCune's automated reasoning program `Otter` (McCune, 1994). The axioms used in this work are a minor modification (Belinfante, 1997) of ones proposed by Quaife (1992a,b), which in turn are based on older work by Boyer et al. (1986) and ultimately on Gödel's (1940) reformulation of the von Neumann–Bernays theory (Bernays, 1958) of classes and sets. To provide an extra challenge, Isbell's (1960) definition of ordinal numbers was used, so that the axiom of regularity does not need to be assumed. We present here an update, as well as describing unanticipated outgrowths of that work.

A prominent feature of Gödel's formalism is the absence of the usual class formation $\{x \mid p(x)\}$ axiom schema. In its place are individual axioms for nine basic class constructors. Definitions of classes must be expressed directly or indirectly in terms of

* Tel.: +1-404-321-6142.
E-mail address: belinfan@math.gatech.edu (J.G.F. Belinfante).

0014-5793/03/$ - see front matter
doi:10.1016/S0747-7171(03)00023-3

these primitives, which include two basic classes, the universal class V and the membership relation E, four unary class constructors complement, domain, flip and rotate, and three binary constructors pairset, cart, and intersection. Gödel (1940) proved a general Class Existence Metatheorem Schema for class formation, whose proof amounts to a recursive algorithm for converting customary definitions of classes using ordinary class formation to expressions built out of the primitive constructors, together with a proof of termination.

Gödel's algorithm was implemented in Mathematica™ (Belinfante, 1996, 2000) to help prepare input files for proofs in set theory using McCune's automated reasoning program Otter. To avoid complicated output, the algorithm was modified to avoid use of Kuratowski's definition for ordered pairs. Gödel's ban on self-membership was also removed since the work on ordinal numbers did not assume the axiom of regularity. Because the likelihood of success in proving theorems using programs like Otter depends critically on the simplicity of the definitions used and the brevity of the statements of the theorems to be proved, over 3000 simplification rules have been added to produce simple output.

Proof summaries and other pertinent information about each of the Otter proofs of the theorems mentioned in this paper, as well as for several thousand other theorems in set theory, may be found on the author's website:

http://www.math.gatech.edu/~belinfan/research/

A recent version of the GOEDEL program is also provided there.

2. Comparing two definitions of ordinal numbers

A class x is said to be *full* (or *transitive*) if every member of x is a subset of x. This condition can be restated succinctly in several equivalent ways, for example as subclass(U(x),x) or as subclass(x,P(x)). Here U(x) is the *sum class* of x, that is, the union of all the members of x, and P(x) is the *power class* of x, the class of all subsets of x. Isbell defined an *ordinal number* as a set with the property that any full proper subset of x is a member of x.

The quantifiers in Isbell's definition can be eliminated by introducing the class FULL of all full sets. Applying Gödel's algorithm to this class identifies it as the complement of the range of the intersection of the membership relation E and the complement of the subset relation S:

```
equal(complement(R(intersection(E,complement(S)))),FULL).
```

Using this class, one can define an ordinal number as a set x satisfying

```
subclass(intersection(FULL,P(x)),succ(x)).
```

Here succ(x) is the *successor* of x, defined as the union of x and the singleton of x. The class OMEGA of all ordinal numbers (according to Isbell's definition) can also be expressed in terms of FULL by a simple formula:

```
equal(complement(image(intersection(complement(Id),
      intersection(S,complement(E))),FULL)),OMEGA).
```

Here `image(x,y)` is the class of all sets `v` for which there is a member `u` of `y` such that `pair(u,v)` belongs to `x`. See Quaife (1992b).

Historically, von Neumann introduced the axiom of regularity to simplify ordinal number theory. When the axiom of regularity is assumed, one can characterize an ordinal number as a full set whose members are full (Monk, 1969). The class of such sets is `intersection(FULL,P(FULL))`. The proof that this class contains Isbell's class `OMEGA` does not use the axiom of regularity. The proof, using `Otter`, that the two classes are equal in the presence of the axiom of regularity required adding only a few new lemmas to Quaife's `RE` group of theorems. To prove Theorem `RE-ON-EQ` all one needs is Lemma `RE-FUL-2` (see Appendix A).

Appendix A lists clauses for theorems involving the axiom of regularity proved using `Otter`. These clauses include a flag `AxReg` that is true or false depending on whether the axiom of regularity holds, thus turning the axiom of regularity into a definition of this flag. The Skolemized version of this definition is:

```
list(usable).
% axiom D:   axiom of regularity
-AxReg | equal(x,0) | member(regular(x),x).                          % AX-RG-1
-AxReg | equal(x,0) | equal(intersection(regular(x),x),0).          % AX-RG-2
AxReg | -equal(IRREG,0).                                            % AX-RG-3
AxReg | -member(x,IRREG)| -equal(intersection(IRREG,x),0).          % AX-RG-4
end_of_list.
```

Introducing the flag `AxReg` does affect the performance of `Otter`, often forcing one to assign a nonzero value to `max_distinct_vars`.

The Skolem function `regular(x)` occurs also in Quaife's work, but he did not deal with the possibility that the axiom of regularity might fail to hold. Reasons for considering such a possibility have been discussed by various authors (Aczel, 1988; Barwise and Etchemendy, 1987). If `AxReg` is false, then there is some class, represented by the Skolem constant `IRREG`, that fails to satisfy the axiom. Theorems about `IRREG` are only of interest if the axiom of regularity does not hold. Theorem `RE-AX-6D` says that the class `OMEGA` is disjoint from the class `IRREG`, and Theorem `RE-AX-6E` says that if `x` is disjoint from `IRREG`, then so is the power class `P(x)`. So also `P(OMEGA)`, `P(P(OMEGA))`, and so on are also disjoint from `IRREG`. The upshot is that there is no obvious way to construct any elements of `IRREG`.

In a theory with proper classes as well as sets one can formulate weaker or stronger axioms depending on whether the quantifiers are chosen to range over sets or classes; the flag `AxReg` refers to the strong axiom of regularity. For the axiom of regularity, the weak and strong versions are in fact equivalent. See for example Rubin (1967). Because of this, one can eliminate the Skolem constant `IRREG` (see theorem `RE-REG-2` in Appendix A).

3. Other formulations of the axiom of regularity

Gödel's algorithm can be used (Belinfante, 2000) to eliminate quantifiers over sets; any statement with quantifiers over sets can be converted to a logically equivalent equation without quantifiers. What happens is that the quantifiers are neatly built into equivalent

set-theoretic constructs like `domain` and `composite`. For example, the axiom of regularity is usually formulated using quantifiers as:

```
AxReg <-> (all x (equal(x,0) | (exists u (member(u,x) & disjoint(u,x))))).
```

Since the quantifier over `u` is restricted to sets, this statement can be converted into an equivalent statement without the existential quantifier:

```
AxReg <-> (all x (equal(0,x) | -disjoint(x,P(complement(x)).
```

The quantifier is now hidden in the introduced power class functor. Replacing `x` by its complement, one finds a compact formulation of the axiom of regularity:

```
AxReg <-> (all x (subclass(P(x),x) -> equal(x,V))).
```

That is, the axiom of regularity says that the universal class `V` is the only class containing its own power class. Both of these reformulations of the axiom of regularity have the advantage over the original one in that their clausifications do not introduce the Skolem function `regular(x)`. Yet another equivalent version of the axiom of regularity with this virtue is:

```
AxReg <-> (all x (subclass(x,image(E,x)) -> equal(x,0)).
```

The author himself likes this formulation best because it makes clear that the intention of the axiom of regularity is to forbid an infinitely descending chain of membership: $\cdots \in x_2 \in x_1 \in x_0$.

4. Proving the consistency of the axiom of regularity

One of the earliest relative consistency proofs in set theory was the proof that the axiom of regularity is consistent with the other axioms of set theory. One can prove this by constructing an inner model of set theory; what is needed is a class `REGULAR` of sets closed under all the basic operations of set theory, such that the axiom of regularity holds when restricted to members of this subclass. Only the broad outlines of the construction of this model will be described here (see Appendix B for a list of theorems proved about the class `REGULAR`).

The basic idea for this construction harks back to a paper by Mirimanoff (1917). One begins by introducing the class `DESCENDING` of all infinitely descending sets `x`, that is, sets with the property that every member of `x` in turn has a member also belonging to `x`. Using Gödel's algorithm, one finds that this condition can be formulated without quantifiers as `subclass(x,image(E,x))`, and that the class of all infinitely descending sets can be defined by

```
equal(complement(fix(composite(E,DISJOINT))),DESCENDING).
```

Here `fix(x)` is the class of fixed points of `x`, and `DISJOINT` is the disjointness relation, defined as

```
equal(composite(Id,complement(composite(E,inverse(E)))),DISJOINT).
```

An *ordinary* or *regular* set is defined as a set which holds no members that belong to an infinitely descending set. For the class REGULAR of regular sets, Gödel's algorithm yields the definition

```
equal(complement(U(DESCENDING)),REGULAR).
```

The GOEDEL program transforms the weak form of the axiom of regularity to the statement subclass(DESCENDING,singleton(0)). It is easy to show that this is equivalent to equal(REGULAR,V), which asserts that all sets are regular.

The class REGULAR is a proper class with many remarkable properties such as being its own power class. Proofs of 75 theorems about REGULAR listed in Appendix B were obtained using Otter. Theorem RE-REG-1 says that the strong form of the axiom of regularity implies the weak form. Theorem RE-REG-2 is the converse. The proof of this theorem depends on a lemma that the transitive closure of any set is a set; the latter assertion is equivalent to the statement equal(U(FULL),V) for which an Otter proof has recently been reported (Belinfante, 2001).

5. Subvariant and invariant subsets

A class y is *invariant* under x if subclass(image(x,y),y) holds. We say that a class y is *subvariant* under x when the reverse inclusion subclass(y,image(x,y)) holds. Using this terminology, an *infinitely descending* set is one that is subvariant under the membership relation E.

The condition of subvariance is useful in many applications as a tool for constructing invariant sets. For example, in the recursion theorem, one constructs a function as a union of approximations that satisfy a subvariance condition. For this reason alone it seems a good idea to embark on a general study of subvariance and its relation to invariance. Over a hundred theorems about invariant and subvariant subsets proved using Otter are listed in Appendix C.

The definitions of the classes of sets that are invariant and subvariant under a given relation x both involve the constructor fix, but they differ crucially:

```
equal(fix(composite(S,IMAGE(x))),invar(x)).
equal(complement(fix(composite(E,complement(
      composite(x,inverse(E)))))),subvar(x)).
```

The function IMAGE(x) maps a set y to image(x,y) whenever the latter is a set. The explanation (Belinfante, 2000) for the difference between these two equations has to do with the distinction between sets and proper classes. The axiom of replacement implies that a subclass of a set is a set. When y is a set, the invariance condition implies that image(x,y) is also a set, and so the ordered pair of x and image(x,y) is a point of the graph of IMAGE(x). But a set y can satisfy the subvariance condition without image(x,y) necessarily being a set. One can get a formula for subvar analogous to the definition of invar if an extra condition holds, for example if all vertical sections of x are sets (see Theorem SBV-IMG3 in Appendix C). The construction of the class of regular sets can be formulated succinctly using subvar as

```
equal(complement(U(subvar(E))),REGULAR).
```

6. The class of finite sets

The customary definition of a finite set as one that can be put into one-to-one correspondence with a natural number makes the concept of finiteness appear to depend on a specific construction of natural numbers. There are various ways to define finiteness without explicit reference to natural numbers (Formisano and Omodeo, 1998; Hrbacek and Jech, 1999; Tarski, 1924). One such idea, adopted here, is to define a set to be finite if it does not belong to an infinitely descending chain of proper subsets. This yields a definition for the class of finite sets formally resembling that of the class of regular sets, but with the proper subset relation `PS` replacing the membership relation `E`:

```
equal(complement(U(subvar(PS))),FINITE).
```

Using this definition of the class `FINITE`, one can prove all the usual theorems about finiteness. Among the theorems about finite sets proved using `Otter` are that all natural numbers are finite, that all other ordinals are infinite, and a key property of `FINITE`, that it is the smallest class which holds the empty set, and is invariant under the operation of adjoining singletons. This property is formally analogous to ordinary induction, so a good name for it would be `FINITE` induction. Applications of `FINITE` induction include the `Otter` proofs obtained for the theorem that the binary union of finite sets is finite, that the power set of a finite set is finite, that a finite set is not equipollent to any proper subset of itself, and that a set is finite if and only if it can be put in one-to-one correspondence with a natural number. A more refined version `FIN-IND-2` of `FINITE` induction was used in the `Otter` proof that the sum class of a finite set of finite sets is finite.

7. Conclusion

For most applications of automated reasoning in modern mathematics the availability of a substantial amount of set theory is essential. Much progress has been made recently toward mechanizing set theory, especially by Larry Paulson and his coworkers (Noël, 1993; Paulson and Grąbczewski, 1996), the Mizar group (Rudnicki and Trybulec, 1999) and Megill (1997), among others. Some of these efforts have been modestly described as proof checking rather than proof finding, but the distinction between the two activities is not sharp, and both are challenging at the present state of the art. I regard it as a healthy sign that each of these groups uses slightly different axioms for set theory, and that the methods employed are generally quite different. Despite all this progress in using computers to find and check the correctness of proofs of theorems in set theory, the process is still far from being routine.

For the work described here, a primary obstacle has been to find succinct and useful definitions of the classes one needs. Starting with the primitive constructors provided by the axioms, the challenge is to introduce just enough additional constructors to help reduce the complexity of the statements of theorems, but not so many that the proliferation of new concepts itself causes an unnecessary explosion in the clause lists. Introducing the notion of subvariance makes possible a unified treatment of regular sets and finite sets, and lays the groundwork for a proof of the recursion theorems needed to develop arithmetic.

Appendix A. Theorems involving the axiom of regularity

A few of Quaife's theorems listed here do not include the flag `AxReg`, and are valid whether or not the axiom of regularity holds.

```
list(usable).
% revised versions of Quaife's Theorems.
-AxReg | equal(intersection(regular(x),x),0).                                    % RE-0
-AxReg | disjoint(regular(x),x).                                                 % RE-0'
-AxReg | -member(x,y) | member(regular(y),y).                                    % RE-0''
-AxReg | -member(x,V) |
     member(regular(union(y,singleton(x))),union(y,singleton(x))).               % RE-0'''
% comment: Quaife's Theorem RE-1 is back-subsumed by his Theorem RE-4
-AxReg | -member(x,x).                                                          % RE-1
-AxReg | -member(x,y) | -subclass(y,x).                                         % RE-1-SU
-AxReg | equal(RUSSELL,V).                                                      % RE-RUS
-AxReg | equal(fix(E),0).                                                       % RE-E-FP
-AxReg | disjoint(E,inverse(S)).                                                % RE-E-SR
-AxReg | -FUNCTION(x) | -subclass(x,composite(inverse(E),x)) | equal(x,0).     % RE-FU
-AxReg | -equal(singleton(x),x).                                                % RE-2
-AxReg | -equal(memb(x),x) | -equal(singleton(memb(x)),x).                      % RE-3
-AxReg | -equal(U(x),x) | -equal(singleton(U(x)),x).                           % RE-3'
-AxReg | -member(x,R(SINGLETON)) | -equal(U(x),x).                             % RE-3''
-AxReg | -member(x,y) | -member(y,x).                                           % RE-4
-AxReg | disjoint(E,inverse(E)).                                               % RE-E-IN
-AxReg | -member(x,U(x)).                                                       % RE-4-A
-AxReg | disjoint(E,BIGCUP).                                                    % RE-E-BC
-AxReg | -member(x,V) | -equal(regular(union(y,singleton(x))),x) | disjoint(y,x).  % RE-4-B1
-AxReg | -member(x,V) | member(regular(union(y,singleton(x))),y) | disjoint(y,x).  % RE-4-B3
-AxReg | -member(x,y) | -member(y,z) | -member(z,x).                            % RE-4-B8
-AxReg | -member(x,U(U(x))).                                                    % RE-4-B9
-AxReg | -member(P(x),U(x)).                                                    % RE-PC-SC
-AxReg | -equal(pair(x,y),x).                                                   % RE-5A
-AxReg | -equal(pair(x,y),y).                                                   % RE-5B
-AxReg | -equal(first(x),x) | -member(x,cart(V,V)).                            % RE-6A
-AxReg | -equal(second(x),x) | -member(x,cart(V,V)).                           % RE-6B
% Quaife's Theorem RE-7 does not require the axiom of regularity
-member(x,V) | -member(complement(x),V).                                        % RE-7
-AxReg | -equal(first(x),x) | equal(second(x),x).                               % RE-8A
-AxReg | -equal(second(x),x) | equal(first(x),x).                               % RE-8B
% Quaife's Theorems RE-9 and RE-10 do not require AxReg.
-equal(pair(first(x),second(x)),x) | member(first(x),V).                        % RE-9A'
-equal(pair(first(x),second(x)),x) | member(second(x),V).                       % RE-9B
-equal(pair(first(x),second(x)),x) | member(x,cart(V,V)).                       % RE-9C'
-equal(pair(first(pair(x,y)),second(pair(x,y))),pair(x,y)) | member(x,V).       % RE-10A
-equal(pair(first(pair(x,y)),second(pair(x,y))),pair(x,y)) | member(y,V).       % RE-10B

% some new theorems
-AxReg | -subclass(U(x),x) | equal(x,0) | member(0,x).                          % RE-FUL-1
-AxReg | -member(P(x),x).                                                       % RE-PC
-AxReg | equal(x,0) | member(regular(x),P(complement(x))).                      % RE-AX-1
-AxReg | equal(x,V) | member(regular(complement(x)),P(x)).                      % RE-AX-2
-AxReg | -disjoint(P(x),complement(x)) | equal(x,V).                           % RE-AX-3
-AxReg | -subclass(P(x),x) | equal(x,V).                                       % RE-AX-4
-AxReg | -full(x) | -subclass(intersection(x,P(y)),y) | subclass(x,y).         % RE-FUL-2

% equivalence of two definitions of ordinal numbers.
-AxReg | equal(intersection(FULL,P(FULL)),OMEGA).                              % RE-ON-EQ
% a property of the class of ordinal numbers
-AxReg | -equal(intersection(FULL,P(x)),x) | equal(x,OMEGA).                   % RE-ON-ON

% Other consequences of the axiom of regularity.
-AxReg | equal(REGULAR,V).                                                     % RE-REG-1
-AxReg | -subclass(x,image(E,x)) | equal(x,0).                                % RE-IM
% sequential formulation of the infinite regress condition
-AxReg | -subclass(omega,D(x)) |
```

```
     -subclass(composite(x,inverse(SUCC)),composite(E,x)).                    % RE-OM
-AxReg | -subclass(x,composite(E,x)) | equal(x,0).                            % RE-CO
-AxReg | equal(fix(composite(E,DISJOINT)),complement(singleton(0))).          % RE-DJT-E
% lemma RE-E-LEM does not require AxReg.
-equal(D(x),0) | -subclass(x,E) | equal(x,0).                                 % RE-E-LEM
-AxReg | -subclass(x,E) | -subclass(D(x),R(x)) | equal(x,0).                  % RE-E
% theorems about IRREG
AxReg | disjoint(P(complement(IRREG)),IRREG).                                 % RE-AX-5
AxReg | -equal(complement(IRREG),V).                                         % RE-AX-6A
AxReg | subclass(P(complement(IRREG)),complement(IRREG)).                    % RE-AX-6B
% the proof of RE-REG-2 uses the lemma U(FULL) = V.
-equal(REGULAR,V) | AxReg.                                                   % RE-REG-2
-member(0,IRREG) | AxReg.                                                    % RE-AX-6C
AxReg | disjoint(OMEGA,IRREG).                                               % RE-AX-6D
-disjoint(x,IRREG) | AxReg | disjoint(P(x),IRREG).                           % RE-AX-6E
AxReg | subclass(IRREG,image(E,IRREG)).                                      % RE-AX-7
-equal(REGULAR,V) | -member(IRREG,V) | AxReg.                                % RE-REG-3
-member(IRREG,REGULAR) | AxReg.                                              % RE-REG-4
end_of_list.
```

Theorem RE-OM says that the axiom of regularity does not permit a sequence of sets with each one holding the next as an element. This clumsy sequential formulation of the ban on infinite regress for membership explicitly mentions the set omega of natural numbers and the successor function SUCC.

Appendix B. Theorems about the class of regular sets

In this appendix are listed clauses for the definitions of the classes of descending and regular sets, and theorems about these proved using Otter. Clauses flagged with an asterisk are equations which are (usually) also placed on the demodulator list in Otter input files.

```
list(usable).
% definition of the class of descending sets
equal(complement(fix(composite(E,DISJOINT))),DESCENDING).                    %*DF-DESC

% theorems about the class DESCENDING
-member(x,DESCENDING) | subclass(x,image(E,x)).                              % DESC-1
-member(x,V) | -subclass(x,image(E,x)) | member(x,DESCENDING).               % DESC-2
member(0,DESCENDING).                                                        % DESC-3
equal(A(DESCENDING),0).                                                      %*DESC-A
-member(x,P(DESCENDING)) | member(U(x),DESCENDING).                          % DESC-4
-member(x,DESCENDING) | member(union(x,singleton(y)),DESCENDING) | disjoint(x,y). % DESC-5
-member(x,y) | -member(x,U(DESCENDING)) | -member(y,V) | member(y,U(DESCENDING)). % DESC-6
subclass(image(E,U(DESCENDING)),U(DESCENDING)).                              % DESC-7
-member(x,x) | member(singleton(x),DESCENDING).                              % DESC-8
-member(x,complement(x)) | -member(singleton(x),DESCENDING).                 % DESC-9
-member(x,y) | -member(y,x) | member(pairset(x,y),DESCENDING).                % DESC-10

% definition of the class of regular sets
equal(complement(U(DESCENDING)),REGULAR).                                    %*DF-REG

% theorems about the class REGULAR
equal(complement(REGULAR),U(DESCENDING)).                                    %*REG-C
% the class REGULAR is its own power class
equal(P(REGULAR),REGULAR).                                                   %*REG-2
equal(image(E,U(DESCENDING)),U(DESCENDING)).                                 %*REG-2COR
-subclass(x,REGULAR) | subclass(U(x),REGULAR).                               % REG2-SU1
-subclass(U(x),REGULAR) | subclass(x,REGULAR).                               % REG2-SU2
-subclass(x,REGULAR) | subclass(P(x),REGULAR).                               % REG2-SU3
equal(image(inverse(S),U(DESCENDING)),image(V,U(DESCENDING))).               % REG-C-SR
```

```
equal(U(U(DESCENDING)),image(V,U(DESCENDING))).                                  % REG-C-SC
-member(DESCENDING,V) | equal(REGULAR,V).                                        % REG-C-V
equal(image(Di,U(DESCENDING)),image(V,U(DESCENDING))).                           % REG-C-DI
equal(A(U(DESCENDING)),complement(image(V,U(DESCENDING)))).                      % REG-C-A
equal(image(V,REGULAR),V).                                                       %*REG-IMV
equal(U(REGULAR),REGULAR).                                                       %*REG-3
-subclass(P(x),REGULAR) | subclass(x,REGULAR).                                   % REG3-SU
equal(image(inverse(S),REGULAR),REGULAR).                                        %*REG-HER
-member(x,REGULAR) | subclass(x,REGULAR).                                        % REG-4
-member(x,y) | -member(y,REGULAR) | member(x,REGULAR).                           % REG-5
-member(x,REGULAR) | member(P(x),REGULAR).                                       % REG-6
-member(x,REGULAR) | member(U(x),REGULAR).                                       % REG-7
-member(x,REGULAR) | -subclass(y,x) | member(y,REGULAR).                         % REG-8
member(0,REGULAR).                                                               % REG-9
equal(intersection(REGULAR,P(U(DESCENDING))),singleton(0)).                      %*REG-C-PC
equal(image(S,REGULAR),V).                                                       %*REG-IMS
equal(image(DISJOINT,REGULAR),V).                                                %*REG-DJT
equal(A(REGULAR),0).                                                             %*REG-A
-member(x,REGULAR) | -member(y,REGULAR) | member(union(x,y),REGULAR).            % REG-10
-member(x,REGULAR) | member(intersection(x,y),REGULAR).                          % REG-11
-member(x,REGULAR) | member(singleton(x),REGULAR).                               % REG-12
-member(x,REGULAR) | -member(y,REGULAR) | member(pairset(x,y),REGULAR).          % REG-13
-member(P(x),REGULAR) | member(x,REGULAR).                                       % REG-14
-member(U(x),REGULAR) | member(x,REGULAR).                                       % REG-15
-member(pair(x,y),cart(REGULAR,REGULAR)) | member(cart(x,y),REGULAR).            % REG-CP
-member(x,DESCENDING) | disjoint(x,REGULAR).                                     % REG-16
-member(x,DESCENDING) | -member(y,REGULAR) | disjoint(x,y).                      % REG-17
equal(intersection(DESCENDING,REGULAR),singleton(0)).                            %*REG-19
-disjoint(DESCENDING,image(E,x)) | subclass(x,REGULAR).                          % REG-20
-subclass(x,REGULAR) | disjoint(DESCENDING,image(E,x)).                          % REG-21
-member(x,REGULAR) | member(D(x),REGULAR).                                       % REG-22
-member(x,REGULAR) | member(R(x),REGULAR).                                       % REG-23
-member(x,REGULAR) | member(rotate(x),REGULAR).                                  % REG-RO
-subclass(x,REGULAR) | subclass(D(x),REGULAR).                                   % REG-24
-subclass(x,REGULAR) | subclass(R(x),REGULAR).                                   % REG-25
subclass(cart(REGULAR,REGULAR),REGULAR).                                         % REG-26
-member(x,REGULAR) | -subclass(x,image(E,x)) | equal(x,0).                       % REG-27
-member(REGULAR,x).                                                              % REG-28
equal(image(Di,REGULAR),V).                                                      %*REG-DI
equal(singleton(REGULAR),0).                                                     %*REG-29
-member(cart(REGULAR,REGULAR),x).                                               % REG-CP-V
equal(D(REGULAR),REGULAR).                                                       %*REG-30
equal(R(REGULAR),REGULAR).                                                       %*REG-31
-member(x,REGULAR) | member(first(x),REGULAR).                                   % REG-FST
-member(x,REGULAR) | member(second(x),REGULAR).                                  % REG-SEC
equal(intersection(REGULAR,cart(V,V)),cart(REGULAR,REGULAR)).                    %*REG-32
equal(composite(x,REGULAR),cart(REGULAR,image(x,REGULAR))).                      %*REG-CO1
equal(composite(REGULAR,x),cart(image(inverse(x),REGULAR),REGULAR)).             % REG-CO2
equal(image(REGULAR,x),intersection(REGULAR,image(V,intersection(REGULAR,x)))).  %*REG-IM
equal(inverse(REGULAR),cart(REGULAR,REGULAR)).                                   %*REG-34
-member(x,REGULAR) | member(inverse(x),REGULAR).                                 % REG-IN
-member(x,REGULAR) | member(flip(x),REGULAR).                                    % REG-FL
equal(fix(REGULAR),REGULAR).                                                     %*REG-FP
equal(intersection(Id,REGULAR),id(REGULAR)).                                     %*REG-IDX
subclass(REGULAR,RUSSELL).                                                       % REG-35

% theorems about REGULAR added to other groups.
-member(complement(REGULAR),V) | equal(REGULAR,V).                               % SP-REG-C
-member(DESCENDING,V) | equal(singleton(0),DESCENDING).                          % SP-DESC
equal(image(BIGCUP,REGULAR),REGULAR).                                           % BC-REG-1
-member(intersection(REGULAR,complement(x)),V) |
     -equal(P(x),x) | subclass(REGULAR,x).                                      % REG-PC
equal(image(inverse(BIGCUP),REGULAR),REGULAR).                                  % BC-REG-2
equal(image(inverse(POWER),REGULAR),REGULAR).                                   % POW-REG1
equal(image(POWER,REGULAR),intersection(REGULAR,R(POWER))).                     % POW-REG2
subclass(image(CART,REGULAR),REGULAR).                                          % CART-REG
equal(IMAGE(REGULAR),cart(P(U(DESCENDING)),singleton(0))).                      % IMG-REG2
```

```
% connections with the class FULL of full sets and the class OMEGA of ordinals
-member(x,FULL) | -subclass(y,image(E,y)) | member(intersection(x,y),DESCENDING).   % FULDESC1
-member(x,FULL) | -subclass(P(y),y) | subclass(intersection(REGULAR,x),y).          % FUL-REG1
-subclass(P(x),x) | subclass(intersection(REGULAR,U(FULL)),x).                      % FUL-REG2
-member(x,FULL) | member(x,DESCENDING) | member(0,x).                               % FULDESC2
-member(x,intersection(FULL,REGULAR)) | equal(x,0) | member(0,x).                  % FUL-REG3
subclass(OMEGA,REGULAR).                                                          % ON-REG
end_of_list.
```

Appendix C. Invariant and subvariant sets

In this appendix are listed clauses for the definitions of the classes of invariant and subvariant sets, and theorems about these that were proved using `Otter`.

```
list(usable).
% definition of the class of invariant subsets
equal(fix(composite(S,IMAGE(x))),invar(x)).                                       %*DEF-IVR

% theorems about invar(x).
-member(x,invar(y)) | subclass(image(y,x),x).                                     % IVR-1
-member(x,y) | -member(y,invar(BIGCUP)) | member(U(x),y).                         % IVR-BC1
-member(x,invar(BIGCUP)) | member(U(x),FULL).                                     % IVR-BC1A
-member(x,invar(BIGCUP)) | member(image(inverse(S),x),FULL).                     % IVR-BC1B
-member(D(x),invar(x)) | subclass(R(x),D(x)).                                    % IVR-DO1
-member(x,V) | -subclass(image(y,x),x) | member(x,invar(y)).                     % IVR-2
-equal(composite(x,y),composite(y,x)) | -member(R(y),V) |
     member(fix(IMAGE(y)),invar(IMAGE(x))).                                      % IVR-FP
-member(x,omega) | member(intersection(omega,complement(x)),invar(SUCC)).        % IVR-OM-C
-member(x,invar(BIGCUP)) | member(image(inverse(S),x),invar(BIGCUP)).           % IVR-BC-S
-member(D(x),V) | -subclass(R(x),D(x)) | member(D(x),invar(x)).                  % IVR-DO2
-member(x,FULL) | member(P(x),invar(BIGCUP)).                                    % IVR-BC2
equal(U(invar(BIGCUP)),U(FULL)).                                                  %*IVR-BC2B
equal(image(inverse(S),invar(BIGCUP)),U(FULL)).                                   %*IVR-BC2C
subclass(intersection(FULL,R(POWER)),invar(BIGCUP)).                              % IVR-BC3
-member(x,invar(y)) | member(image(y,x),invar(y)).                               % IVR-IM
-member(D(x),invar(x)) | member(R(x),invar(x)).                                  % IVR-RA
-subclass(composite(x,y),composite(y,z)) |
     subclass(image(IMAGE(y),invar(z)),invar(x)).                                % IVR-IMG1
subclass(image(IMAGE(x),invar(x)),invar(x)).                                      % IVR-IMG2
-equal(D(IMAGE(x)),V) | subclass(image(BIGCUP,invar(IMAGE(x))),invar(x))         % IVR-IMG3
-subclass(x,y) | subclass(invar(y),invar(x)).                                    % IVR-SU
equal(invar(union(x,y)),intersection(invar(x),invar(y))).                         %*IVR-U
subclass(P(complement(D(x))),invar(x)).                                           % IVR-C-DO
-member(D(x),V) | -member(invar(x),V).                                           % IVR-V
member(0,invar(x)).                                                               % IVR-MEM0
equal(A(invar(x)),0).                                                             %*IVR-0-A
subclass(invar(x),D(IMAGE(x))).                                                   % IVR-3
equal(fix(composite(complement(E),composite(x,inverse(E)))),complement(invar(x))). %*IVR-5
-member(x,P(invar(y))) | member(U(x),invar(y)).                                  % IVR-SC1
equal(image(BIGCUP,P(invar(x))),invar(x)).                                        %*IVR-SC1A
-subclass(x,invar(y)) | subclass(image(y,U(x)),U(x)).                           % IVR-SC2
-subclass(x,invar(y)) | equal(x,0) | member(A(x),invar(y)).                     % IVR-A
equal(image(BIGCAP,P(invar(x))),invar(x)).                                        %*IVR-BA

% some examples
equal(invar(0),V).                                                                %*IVR-0
equal(invar(Id),V).                                                               %*IVR-ID
equal(invar(id(x)),V).                                                            %*IVR-IDX
equal(invar(E),singleton(0)).                                                     %*IVR-E
equal(invar(S),singleton(0)).                                                     %*IVR-SR
equal(invar(inverse(S)),fix(IMAGE(inverse(S)))).                                  %*IVR-HER1
subclass(fix(IMAGE(inverse(S))),invar(inverse(SUCC))).                            % IVR-HER2
equal(invar(inverse(E)),FULL).                                                    %*IVR-FUL1
subclass(image(BIGCUP,invar(BIGCUP)),FULL).                                       % IVR-FUL2
```

```
subclass(FULL,invar(inverse(SUCC))).                                          % IVR-FUL3
equal(image(BIGCUP,P(FULL)),FULL).                                            %*IVR-FUL4
equal(image(BIGCAP,P(FULL)),FULL).                                            %*IVR-FUL5

% applications to ordinal number theory
member(omega,invar(SUCC)).                                                    % IVR-OM1
subclass(OMEGA,invar(BIGCUP)).                                                % IVR-ON-1
equal(intersection(P(omega),invar(BIGCUP)),succ(omega)).                      %*IVR-OM2
subclass(intersection(OMEGA,fix(BIGCUP)),invar(SUCC)).                        % IVR-ON-2
equal(intersection(OMEGA,invar(SUCC)),intersection(OMEGA,fix(BIGCUP))).       %*IVR-ON3A
subclass(OMEGA,union(R(SUCC),invar(SUCC))).                                   % IVR-ON-4
disjoint(intersection(R(SUCC),invar(SUCC)),RUSSELL).                          % IVR-SUC
equal(intersection(image(SUCC,OMEGA),invar(SUCC)),0).                         %*IVR-ON-5
equal(intersection(fix(BIGCUP),image(SUCC,OMEGA)),0).                         %*IVR-ON-6

% definition of the class of subvariant subsets
equal(complement(fix(composite(E,complement(composite(x,inverse(E))))),subvar(x)). %*DEF-SBV

% some examples
equal(subvar(composite(Id,x)),subvar(x)).                                     %*SBV-IDC0
equal(subvar(composite(id(x),y)),intersection(P(x),subvar(y))).               %*SBV-RS
equal(subvar(0),singleton(0)).                                                %*SBV-0
equal(subvar(V),V).                                                           %*SBV-V
equal(subvar(Id),V).                                                          %*SBV-ID
equal(subvar(id(x)),P(x)).                                                    %*SBV-IDX
equal(subvar(Di),complement(R(SINGLETON))).                                   %*SBV-DI
equal(subvar(cart(V,x)),P(x)).                                                %*SBV-CP-V
equal(subvar(E),DESCENDING).                                                  %*SBV-E
equal(subvar(inverse(BIGCUP)),invar(BIGCUP)).                                 %*SBV-BC

% basic theorems about subvar
-member(x,subvar(y)) | subclass(x,image(y,x)).                                % SBV-1
-member(x,V) | -subclass(x,image(y,x)) | member(x,subvar(y)).                 % SBV-2
% technical lemmas needed for the theory of finite sets
-member(x,y) | -member(y,subvar(PS)) | member(intersection(y,P(x)),subvar(PS)). % SBV-PS1
-member(x,y) | -subclass(y,image(PS,y)) | member(intersection(y,P(x)),subvar(PS)). % SBV-PS1'
-member(x,subvar(PS)) | -subclass(y,A(x)) |
     member(image(IMAGE(id(complement(y))),x),subvar(PS)).                    % SBV-PS2
-member(x,V) | -member(0,y) | -subclass(image(CUP,cart(y,R(SINGLETON))),y) |
     member(intersection(P(x),complement(y)),subvar(PS)).                     % SBV-PS3
-member(x,V) | -member(0,y) | -subclass(image(CUP,cart(y,image(SINGLETON,x))),y) |
     member(intersection(P(x),complement(y)),subvar(PS)).                     % SBV-PS3A
-member(0,x) | -subclass(image(CUP,cart(x,image(SINGLETON,y))),x) |
     subclass(intersection(P(y),complement(x)),
     image(PS,intersection(P(y),complement(x)))).                             % SBV-PS3B
-member(x,V) | -member(0,y) | -subclass(image(K,y),y) |
     member(intersection(P(x),complement(y)),subvar(PS)).                     % SBV-PS3K
-member(x,FUNS) | -member(y,intersection(subvar(PS),P(P(R(x))))) |
     member(image(IMAGE(inverse(x)),y),subvar(PS)).                           % SBV-PS4
member(intersection(P(omega),complement(image(inverse(S),omega))),subvar(PS)). % SBV-OMPS

% other basic theorems about subvar
-member(x,P(subvar(y))) | member(U(x),subvar(y)).                             % SBV-PC
-member(pair(x,y),cart(subvar(z),subvar(u))) |
     member(cart(x,y),subvar(cross(z,u))).                                    % SBV-X
subclass(intersection(subvar(x),subvar(y)),subvar(composite(x,y))).           % SBV-C0
-subclass(x,y) | subclass(subvar(x),subvar(y)).                               % SBV-SU1
subclass(subvar(x),P(R(x))).                                                  % SBV-SU2
-member(x,V) | member(subvar(x),V).                                           % SBV-MEM
subclass(P(fix(x)),subvar(x)).                                                % SBV-SU3
equal(subvar(S),V).                                                           %*SBV-SR1
equal(subvar(inverse(S)),V).                                                  %*SBV-SR2
equal(subvar(Q),V).                                                           %*SBV-Q
member(0,subvar(x)).                                                          % SBV-MEM0
member(0,image(IMAGE(x),subvar(y))).                                          % SBV-IM-0
equal(A(subvar(x)),0).                                                        %*SBV-A
-member(x,subvar(y)) | -member(z,image(y,x)) |
```

```
     member(union(x,singleton(z)),subvar(y)).                                  % SBV-SS
equal(image(x,U(subvar(x))),U(subvar(x))).                                     %*SBV-SC2

% using subvar to construct invariant subsets
-member(x,FUNS) | member(U(subvar(inverse(x))),invar(x)).                      % SBV-SC3
subclass(invar(POWER),subvar(inverse(E))).                                     % SBV-E-IN
subclass(intersection(P(D(x)),invar(x)),subvar(inverse(x))).                   % SBV-IVR1
subclass(intersection(P(R(x)),invar(inverse(x))),subvar(x)).                   % SBV-IVR2
equal(intersection(invar(x),subvar(x)),fix(IMAGE(x))).                         %*SBV-IVR4
equal(intersection(FULL,subvar(inverse(E))),fix(BIGCUP)).                      %*SBV-FUL

% special results for functions
-FUNCTION(x) | subclass(subvar(inverse(x)),invar(x)).                          % SBV-FU1
-FUNCTION(x) | equal(subvar(inverse(x)),intersection(P(D(x)),invar(x))).       % SBV-FU2
equal(subvar(SWAP),fix(IMAGE(SWAP))).                                          %*SBV-SW
equal(D(intersection(S,IMAGE(x))),intersection(D(IMAGE(x)),subvar(x))).        % SBV-IMG2
equal(fix(composite(inverse(S),IMAGE(x))),intersection(D(IMAGE(x)),subvar(x))). %*SBV-IMG3
-member(x,intersection(D(IMAGE(y)),subvar(y))) | member(image(y,x),subvar(y)). % SBV-IMG4
subclass(image(IMAGE(x),subvar(x)),subvar(x)).                                 % SBV-IMG5
-subclass(composite(x,y),composite(z,x)) |
     subclass(image(IMAGE(x),subvar(y)),subvar(z)).                            % SBV-IMG6
subclass(image(x,U(intersection(D(IMAGE(x)),subvar(x)))),U(subvar(x))).        % SBV-SC4

% building in initial conditions
-member(x,subvar(union(id(y),z))) | subclass(x,union(y,image(z,x))).           % SBV-U-1
-member(x,V) | -subclass(x,union(y,image(z,x))) |
     member(x,subvar(union(id(y),z))).                                         % SBV-U-2
-member(x,intersection(D(IMAGE(y)),subvar(union(id(z),y)))) |
     member(union(x,image(y,x)),subvar(union(id(z),y))).                       % SBV-U-3
-member(R(x),V) | member(subvar(x),V).                                         % SBV-RA1
-member(R(x),V) | member(U(subvar(x)),invar(x)).                               % SBV-RA2
-member(R(x),V) | member(U(subvar(x)),fix(IMAGE(x))).                          % SBV-RA3
subclass(subvar(SUCC),DESCENDING).                                             % SBV-SUC1
equal(intersection(REGULAR,subvar(SUCC)),singleton(0)).                        %*SBV-SUC2
equal(intersection(P(OMEGA),subvar(SUCC)),singleton(0)).                       %*SBV-SUC3
end_of_list.
```

Appendix D. Theorems about the class of finite sets

In this appendix are listed clauses for the definition of the class of finite sets, and theorems about finite sets proved using `Otter`. The relation `Q` is the equipollence relation, and the relation `K` is the cover relation: `pair(x,y)` belongs to `K` if `y` has exactly one more element than `x`. The function `CUP` takes `pair(x,y)` to `union(x,y)`.

```
list(usable).
% definition of the class of finite sets
equal(complement(U(subvar(PS))),FINITE).                                       % DEF-FIN

% theorems about the class FINITE
equal(U(subvar(PS)),complement(FINITE)).                                       %*FIN-C-DF
equal(image(PS,complement(FINITE)),complement(FINITE)).                        %*FIN-PS1

% three versions of the FINITE induction theorem
-member(0,x) | -subclass(image(CUP,cart(x,R(SINGLETON))),x) | subclass(FINITE,x). % FIN-IND1
-member(0,x) | -subclass(image(K,x),x) | subclass(FINITE,x).                   % FIN-K-1
-member(0,x) | -subclass(image(CUP,cart(x,image(SINGLETON,y))),x) |
     subclass(intersection(FINITE,P(y)),x).                                    % FIN-IND2
-member(x,subvar(PS)) | disjoint(x,FINITE).                                    % FIN-DJ1

% the empty set is finite
member(0,FINITE).                                                              % FIN-0
equal(A(FINITE),0).                                                            %*FIN-A
% all singletons are finite
```

```
member(singleton(x),FINITE).                                                    % FIN-SS
equal(image(BIGCUP,FINITE),V).                                                  %*FIN-BC1
subclass(R(SINGLETON),FINITE).                                                  % FIN-RASG
equal(image(BIGCUP,P(FINITE)),V).                                               %*FIN-BC2
equal(U(FINITE),V).                                                             %*FIN-SC1
-member(FINITE,x).                                                              % FIN-MEM
% the set of natural numbers is not finite
-member(omega,FINITE).                                                          % FIN-C-OM
-member(x,complement(FINITE)) | member(x,U(intersection(P(P(x)),subvar(PS)))).  % FIN-PP1

% a function with a finite domain has a finite range
-member(x,FUNS) | -member(D(x),FINITE) | member(R(x),FINITE).                   % FIN-FS
-equal(intersection(P(P(x)),subvar(PS)),singleton(0)) | -member(x,V) |
     member(x,FINITE).                                                          % FIN-PP2
% subsets of finite sets are finite
-member(x,FINITE) | -subclass(y,x) | member(y,FINITE).                         % FIN-SU
% images of finite sets under functions are finite
-FUNCTION(x) | -member(y,FINITE) | member(image(x,y),FINITE).                  % FIN-FU1
-FUNCTION(x) | subclass(image(IMAGE(x),FINITE),FINITE).                        % FIN-FU2
-member(P(x),FINITE) | member(x,FINITE).                                       % FIN-PC1
-member(x,FINITE) | member(D(x),FINITE).                                       % FIN-DO1
-member(x,FINITE) | member(R(x),FINITE).                                       % FIN-RA1
-member(x,FINITE) | member(inverse(x),FINITE).                                 % FIN-IN1
-member(x,FINITE) | member(fix(x),FINITE).                                     % FIN-FP1
-member(x,FINITE) | member(id(x),FINITE).                                      % FIN-IDX
-FUNCTION(x) | -member(y,FINITE) | member(composite(x,y),FINITE).              % FIN-CO1
-FUNCTION(x) | -member(y,FINITE) | member(composite(y,inverse(x)),FINITE).     % FIN-CO2
-member(x,FINITE) | -member(y,subvar(PS)) | disjoint(y,P(x)).                  % FIN-DJ2
-subclass(x,image(PS,x)) | disjoint(FINITE,x).                                 % FIN-DJ3
-member(U(x),FINITE) | member(x,P(FINITE)).                                    % FIN-SC2
-member(x,FINITE) | equal(intersection(P(P(x)),subvar(PS)),singleton(0)).      % FIN-PP3
-member(x,y) | -member(z,intersection(P(P(y)),subvar(PS))) |
     -member(intersection(y,complement(singleton(x))),FINITE) | member(x,A(z)).  % FIN-PP4
-member(intersection(complement(singleton(x)),y),FINITE) | member(y,FINITE).    % FIN-C-SS

% adding one more element to a finite set yields a finite set
-member(x,FINITE) | member(union(x,singleton(y)),FINITE).                      % FIN-U-SS
member(pairset(x,y),FINITE).                                                    % FIN-UP
member(pair(x,y),FINITE).                                                       % FIN-OP

% various demodulators
equal(composite(FINITE,x),cart(D(x),V)).                                        %*FIN-CO3
equal(composite(x,FINITE),cart(V,R(x))).                                        %*FIN-CO4
equal(inverse(FINITE),cart(V,V)).                                               %*FIN-IN2
equal(D(FINITE),V).                                                             %*FIN-DO2
equal(R(FINITE),V).                                                             %*FIN-RA2
equal(fix(FINITE),V).                                                           %*FIN-FP2
equal(image(FINITE,x),image(V,x)).                                              %*FIN-IMV
subclass(cart(x,y),FINITE).                                                     % FIN-CP1
subclass(image(CUP,cart(FINITE,R(SINGLETON))),FINITE).                          % FIN-CUP1
-member(x,V) | member(intersection(P(x),complement(FINITE)),subvar(PS)).       % FIN-PC2
-member(x,FINITE) | subclass(FINITE,image(inverse(ADJOIN(x)),FINITE)).         % FIN-ADJ

% the union of two finite sets is finite
-member(pair(x,y),cart(FINITE,FINITE)) | member(union(x,y),FINITE).             % FIN-U
equal(image(CUP,cart(FINITE,FINITE)),FINITE).                                   %*FIN-CUP2
% the union of a finite collection of finite sets is finite
-member(x,FINITE) | -subclass(x,FINITE) | member(U(x),FINITE).                 % FIN-SC4
equal(image(BIGCUP,intersection(FINITE,P(FINITE))),FINITE).                     %*FIN-SC5
-member(P(x),FINITE) | member(P(union(x,singleton(y))),FINITE).                % FIN-PCSS
subclass(FINITE,image(inverse(POWER),FINITE)).                                  % FIN-POW

% the power class of a finite set is finite
-member(x,FINITE) | member(P(x),FINITE).                                       % FIN-PC3
-member(x,FINITE) | member(subvar(x),FINITE).                                  % FIN-SBV
-member(U(x),FINITE) | member(x,FINITE).                                       % FIN-SC6
equal(image(inverse(BIGCUP),FINITE),intersection(FINITE,P(FINITE))).            %*FIN-SC7
```

```
% cartesian products of finite sets are finite
-member(pair(x,y),cart(FINITE,FINITE)) | member(cart(x,y),FINITE).                % FIN-CP2
-FUNCTION(x) | -member(D(x),FINITE) | member(x,FINITE).                           % FIN-FU3

% a set equipollent to a finite set is finite
equal(image(Q,FINITE),FINITE).                                                   %*FIN-Q-1

% natural numbers are finite
subclass(omega,FINITE).                                                          % FIN-OM2
% the finite sets are those equipollent to a natural number
equal(image(Q,omega),FINITE).                                                    %*FIN-Q-OM
% a characterization of finite sets of natural numbers
equal(image(inverse(S),omega),intersection(FINITE,P(omega))).                    %*FIN-OM3
equal(U(complement(FINITE)),V).                                                  %*FINC-SC
equal(image(BIGCUP,complement(FINITE)),complement(FINITE)).                      %*FIN-BC4
equal(image(inverse(S),FINITE),FINITE).                                          %*FIN-HER
equal(image(inverse(K),FINITE),FINITE).                                          % FIN-K-IN
equal(image(K,FINITE),intersection(FINITE,complement(singleton(0)))).            % FIN-K-3
% the natural numbers are the finite ordinals
equal(intersection(FINITE,OMEGA),omega).                                         %*FIN-ON2
-member(complement(FINITE),x).                                                   % FINC-MEM

% definition of the class of Dedekind finite sets
equal(complement(fix(composite(Q,PS))),DEDEKIND).                                % DEF-DK

% theorems about Dedekind finite sets
-member(omega,DEDEKIND).                                                         % DK-C-OM
equal(fix(composite(Q,PS)),complement(DEDEKIND)).                                % DK-C-DF
% characterization of Dedekind finiteness
-member(x,DEDEKIND) | -member(pair(x,y),Q) | -subclass(x,y) | equal(x,y).        % DK-MEM-1
-member(x,DEDEKIND) | -member(pair(x,y),Q) | -subclass(y,x) | equal(x,y).        % DK-MEM-2

% pigeon-hole principle
-ONEONE(x) | -member(D(x),DEDEKIND) | -subclass(R(x),D(x)) | equal(R(x),D(x)).   % DK-PIG
subclass(image(K,DEDEKIND),DEDEKIND).                                            % DK-K
subclass(image(inverse(K),DEDEKIND),DEDEKIND).                                   % DK-K-IN
equal(image(Q,DEDEKIND),DEDEKIND).                                               % DK-Q
member(0,DEDEKIND).                                                              % DK-0
% finite sets are Dedekind finite
subclass(FINITE,DEDEKIND).                                                       % DK-FIN
% natural numbers are equipollent if and only if they are equal
equal(intersection(Q,cart(omega,omega)),id(omega)).                              % DK-OM-Q2
end_of_list.
```

References

Aczel, P., 1988. Non-Well-Founded Sets, CSLI Lecture Notes, vol. 14. Center for the Study of Language and Information, Stanford.

Barwise, J., Etchemendy, J., 1987. The Liar: An Essay on Truth and Circular Propositions. Oxford University Press, Oxford.

Belinfante, J.G.F., 1996. On a modification of Gödel's algorithm for class formation. Association for Automated Reasoning News Letter (34), 10–15.

Belinfante, J.G.F., 1997. On Quaife's development of class theory. Association for Automated Reasoning Newsletter (37), 5–9.

Belinfante, J.G.F., 1999a. Computer proofs in Gödel's class theory with equational definitions for composite and cross. Journal of Automated Reasoning 22, 311–339.

Belinfante, J.G.F., 1999b. On computer-assisted proofs in ordinal number theory. Journal of Automated Reasoning 22, 341–378.

Belinfante, J.G.F., 2000. Gödel's algorithm for class formation. In: McAllester, D. (Ed.), Automated Deduction—CADE 17. Proceedings of the 17th International Conference on Automated

Deduction Held in Pittsburgh, Pa, USA, June 2000. Springer Verlag Lecture Notes in Artificial Intelligence, vol. 1831, Berlin, pp. 132–147.

Belinfante, J.G.F., (19–23 June 2001). Computer Proofs about Transitive Closure, pp. 11–20. International Joint Conference on Automated Reasoning. IJCAR-2001 Short Papers. Goré, R., Leitsch, A., Nipkow, T. (Eds.), Technical Report DII 11/01, Siena, Italy.

Bernays, P., 1958. Axiomatic Set Theory, North Holland Publishing Co., Amsterdam (first ed.: 1958; second ed.: 1968). Republished 1991 by Dover Publications, New York.

Boyer, R., Lusk, E., McCune, W., Overbeek, R., Stickel, M., Wos, L., 1986. Set theory in first order logic: clauses for Gödel's axioms. Journal of Automated Reasoning 2, 287–327.

Formisano, A., Omodeo, E.G., 1998. An equational re-engineering of set theories, Presented at the FTP'98 International Workshop on First Order Theorem Proving (November 23–25, 1998).

Gödel, K., 1940. The Consistency of the Axiom of Choice and of the Generalized Continuum Hypothesis with the Axioms of Set Theory. Princeton University Press, Princeton.

Hrbacek, K., Jech, T., 1999. Introduction to Set Theory, third ed., Revised and Expanded. Marcel Dekker, New York.

Isbell, J.R., 1960. A definition of ordinal numbers. The American Mathematical Monthly 67, 51–52.

McCune, W.W., January 1994. Otter 3.0 Reference Manual and Guide, Argonne National Laboratory Report ANL–94/6, Argonne National Laboratory, Argonne, IL.

Megill, N.D., 1997. Metamath: A Computer Language for Pure Mathematics.

Mirimanoff, D., 1917. The antinomies of Russell and of Burali–Forte and the fundamental problem of the theory of sets. L'enseignment mathematique 19, 209–217 (in French).

Monk, J.D., 1969. Introduction to Set Theory. McGraw-Hill, New York.

Noël, P.A.J., 1993. Experimenting with Isabelle in ZF set theory. Journal of Automated Reasoning 10, 15–58.

Paulson, L.C., Grąbczewski, K., 1996. Mechanizing set theory. Journal of Automated Reasoning 17, 291–323.

Quaife, A., 1992a. Automated deduction in von Neumann–Bernays–Gödel set theory. Journal of Automated Reasoning 8, 91–147.

Quaife, A., 1992b. Automated Development of Fundamental Mathematical Theories. Ph.D. Thesis, University of California at Berkeley, Kluwer Academic Publishers, Dordrecht.

Rubin, J.E., 1967. Set Theory for the Mathematician. Holden-Day, San Francisco.

Tarski, A., 1924. On finite sets. Fundamenta Mathematicae 6, 45–95 (in French).

Rudnicki, P., Trybulec, A., 1999. On equivalents of well-foundedness. Journal of Automated Reasoning 23, 197–234.